

# Ordered Navigation on Multi-attributed Data Words<sup>\*</sup>

Normann Decker<sup>1</sup>, Peter Habermehl<sup>2</sup>, Martin Leucker<sup>1</sup>, and Daniel Thoma<sup>1</sup>

<sup>1</sup> ISP, University of Lübeck, Germany  
`{decker,leucker,thoma}@isp.uni-luebeck.de`

<sup>2</sup> Univ Paris Diderot, Sorbonne Paris Cité, LIAFA, CNRS, France  
`peter.habermehl@liafa.univ-paris-diderot.fr`

**Abstract.** We study temporal logics and automata on multi-attributed data words. Recently, BD-LTL was introduced as a temporal logic on data words extending LTL by navigation along positions of single data values. As allowing for navigation wrt. tuples of data values renders the logic undecidable, we introduce ND-LTL, an extension of BD-LTL by a restricted form of tuple-navigation. While complete ND-LTL is still undecidable, the two natural fragments allowing for either future or past navigation along data values are shown to be Ackermann-hard, yet decidability is obtained by reduction to nested multi-counter systems. To this end, we introduce and study nested variants of data automata as an intermediate model simplifying the constructions. To complement these results we show that imposing the same restrictions on BD-LTL yields two 2ExpSpace-complete fragments while satisfiability for the full logic is known to be as hard as reachability in Petri nets.

## 1 Introduction

Executions of object-oriented and concurrent systems can naturally be modeled using data words. They are composed of labels from a finite alphabet together with a data value from an infinite domain. They can, for example, be considered as an interleaving of actions of an unbounded number of objects or processes, distinguished by identifiers. Recently, several formalisms based on first-order logic [1,2] or temporal logic [3,4,5] have been proposed to specify properties over data words. Automata-based models have also been considered [6,7,8,9] including data automata (DA) [1]. Usually, in these formalisms the data values can only be compared with respect to equality. More expressive relations like ordering lead fast to undecidability. The automata/logic connection has been studied extensively. For example, the satisfiability problem of two-variable first-order logic over data words was shown decidable by a reduction to the emptiness problem of DA [1]. They consist of a finite-state letter-to-letter transducer  $\mathcal{A}$  and a class automaton  $\mathcal{B}$ .  $\mathcal{A}$  changes the labels from the finite alphabet of the input data word before the data word is projected into class strings (one for each different data value) which must all be accepted by  $\mathcal{B}$ . Emptiness of DA was proven decidable by a reduction to the reachability problem in multi-counter systems (Petri nets, VASS)

---

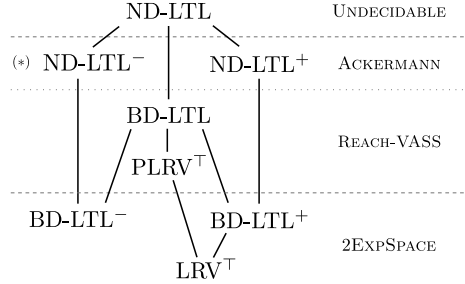
<sup>\*</sup> This work is partially supported by EGIDE/DAAD-Procope (LeMon).

showing a deep connection between data word formalisms and counter systems (see also [5,10,11]).

We study *multi-attributed data words* where, instead of one data value, *several* data values are associated to a given position. This important extension allows for example modeling *nested* parameterized systems where a process has subprocesses which have subprocesses and so on. We built on the logic on multi-attributed data words *basic data LTL* (BD-LTL) [12] allowing for navigation wrt. a data value. It uses the well-known LTL with past-time operators and has additionally a class quantifier over *one data value* used to bind a current data value and restrict the evaluation of the formula to the positions where the same data value appears. Decidability of the satisfiability problem was shown using a reduction to non-emptiness of DA. Adding a class quantifier over tuples makes BD-LTL undecidable like other logics over multi-attributed data words with tuple navigation [5,11].

*Contributions.* We consider first two fragments of BD-LTL: the *class future fragment* BD-LTL<sup>+</sup> (past operators are disallowed for navigation wrt. a data value) and the *class past fragment* BD-LTL<sup>-</sup> (restriction of future operators). Both fragments are shown 2EXPSpace-complete using [5] and revisiting the translation from BD-LTL to DA [12]. Instead of going to general DA we translate BD-LTL<sup>+</sup> and BD-LTL<sup>-</sup> into pDA and sDA, respectively, whose emptiness problems are in EXPSpace. In pDA (resp. sDA) the language of the class automaton is suffix- (resp. prefix-) closed allowing to use the EXPSpace-complete coverability problem of multi-counter systems instead of its reachability problem for which no primitive recursive algorithm is known (cf. [13]). We consider both finite and infinite word semantics of the fragments.

We then define the new logic ND-LTL allowing for navigation wrt. tuples respecting a certain tree-order, i. e., there are several layers of data with nested access. For example, one can navigate on the first layer and, fixing a value, navigate on the second (see example below). Independent navigation on the whole second layer is not possible. While even with this restricted navigation ND-LTL is undecidable we obtain, as for BD-LTL, two natural fragments ND-LTL<sup>+</sup> and ND-LTL<sup>-</sup>. We can prove their decidability by a translation into *nested data automata* (NDA) that we introduce as an appropriate extension of DA. *k*-NDA have *k* class automata and accept data words with *k* data values at each position. The *i*-th class automaton must accept all class strings obtained by projection of the data word using the same *first i* data values. Emptiness of *k*-NDA is undecidable, but shown decidable for *k*-sNDA (where class automata have



**Fig. 1.** Overview of the logics studied in this paper. Lines are drawn downwards to logics with lower expressiveness. The depicted complexity classes apply over finite as well as infinite words except for ND-LTL<sup>-</sup>, marked by (\*), which is undecidable over infinite words.

suffix-closed languages) and  $k$ -pNDA (prefix-closed) using *nested multi-counter systems* (similar to models in [11,14]) which generalize multi-counter systems to several layers of nested counters. Their emptiness problem is undecidable, but, as they are well-structured transition systems [15,16], coverability and control state reachability are decidable.  $\text{ND-LTL}^+$  and  $\text{ND-LTL}^-$  are shown ACKERMANN-hard via a reduction from the control state reachability problem of reset multi-counter systems [17]. Finally,  $\text{ND-LTL}^+$  is decidable over infinite words but  $\text{ND-LTL}^-$  is not. Figure 1 summarizes some results.

*Related Work.* The logics  $\text{LRV}^\top$  (based on [18]) and the more expressive LRV over multi-attributed data words studied in [5] built also on LTL and allow to state that *one* of the current data values must be seen again in the future. LRV ( $\text{LRV}^\top$ ) can be extended to PLRV ( $\text{PLRV}^\top$ ) with past obligations.  $\text{PLRV}^\top$  is less expressive than  $\text{BD-LTL}^3$  and we show that  $\text{LRV}^\top$  is less expressive than  $\text{BD-LTL}^+$ . LRV (and  $\text{LRV}^\top$ ) are  $2\text{EXPSPACE}$ -complete like  $\text{BD-LTL}^+$ . We use their hardness result for our logic. The proof of the upper bound is also based on the coverability problem of multi-counter systems. However, our proof is split into smaller, structured parts. The handling of infinite word versions of our fragments is similar to theirs but we have to treat the additional problems coming from the nested data. Navigation wrt. data tuples was considered and shown undecidable but no decidable fragments were given. A logic handling data values in a very natural way is Freeze-LTL [4]. It exhibits a similar future-restriction as  $\text{BD-LTL}^+$  and  $\text{ND-LTL}^+$  and finite satisfiability is decidable and ACKERMANN-hard. However, satisfiability over infinite words is undecidable while it is still decidable for  $\text{BD-LTL}^+$  and  $\text{ND-LTL}^+$ . In [11], words with nested data values were also considered. They show undecidability for the two-variable logic with two layers of nested data and the  $+1$  and  $<$  predicates over positions. They introduce *higher-order multi-counter automata*, a very similar model to our nested multi-counter systems. Their proof of Turing completeness could be easily adapted to nested multi-counter systems. However, the well-structuredness of the model is not exploited. If the  $+1$  predicate is dropped they obtain decidability, which is orthogonal to our result as we can express the successor relation in our fragments. In [10] history register automata (HRA) have been introduced, which can easily be simulated by our pNDA. A weak variant of HRA is defined which is similar to our pDA, but only studied over finite words.

*Example.* In object-oriented programming languages, iterating over a list is usually done using a method *next* on a corresponding *iterator object*. Once the state of a list changes, e. g., by adding an element, any iterator for that list created before should no longer be used. We model this scenario using propositions *newItr*, *add*, *next* and data words with ordered attributes  $l < s < i$  for identifying the list, the list's state and the iterator, respectively. Thus, fixing a state, fixes also the list it belongs to and fixing an iterator object fixes the corresponding list in its current state.

---

<sup>3</sup> In [5] it is stated without proof that PLRV is also less expressive than  $\text{BD-LTL}$ .

Consider two constraints: (1) When observing *add*, the state of the list changes, i. e., we observe a *fresh* state ID. (2) When calling *next*, the state ID must not have changed since the creation of the currently used iterator. By  $G(\text{add} \rightarrow C_s \neg Y^\perp \top)$  we can express (1). We bind the current state and list IDs using a class quantifier  $C_s$  and check that there is no previous position with the same IDs. We express (2) by  $G(\text{next} \rightarrow C_i(\top S^\perp \text{newItr}))$ .  $C_i$  binds the current state, list and iterator ID. The formula  $(\top S^\perp \text{newItr})$  guarantees that there is a previous position with the same IDs where the iterator is created. For both constraints we use  $\text{ND-LTL}^-$  with local past operators ( $Y^\perp$  and  $S^\perp$ ).

## 2 Preliminaries

Let  $\mathbb{N} = \{0, 1, 2, \dots\}$  be the set of natural numbers and  $[k] := \{1, \dots, k\}$  for  $k \in \mathbb{N}, k > 0$ . We denote the set of finite words over an alphabet  $\Sigma$  by  $\Sigma^*$ , the set of infinite words by  $\Sigma^\omega$  and their union by  $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ . The empty word is denoted  $\epsilon$ . The *shuffle* of two words  $w, w' \in \Sigma^\infty$  is inductively defined by  $\epsilon \sqcup w = w \sqcup \epsilon = \{w\}$  and  $aw \sqcup a'w' = a(w \sqcup a'w') \cup a'(aw \sqcup w')$  where  $a, a' \in \Sigma$ . The shuffle of two languages  $L, L' \subseteq \Sigma^\infty$  is  $L \sqcup L' = \{w \sqcup w' \mid w \in L, w' \in L'\}$  and  $\sqcup(L) = \bigcup \{M \mid M \subseteq L \sqcup M\}$  denotes the infinite shuffle of a language with itself. For two sets  $M, N$  we denote by  $M^N$  the set of all mappings  $f : N \rightarrow M$  from  $N$  to  $M$ . Given a partial order  $(M, \sqsubseteq)$  we write  $m^\downarrow = \{m' \in M \mid m' \sqsubseteq m\}$  for the downward closure of  $m \in M$ . We define a *tree order*  $(M, \leq)$  to be a partial order s. t. for all  $m \in M$  its downward closure is a linear order  $(m^\downarrow, \leq)$ . Hence, we allow a tree order to contain several minimal elements (roots).

An  $\infty$ -automaton over a finite *input alphabet*  $\Sigma$  is a tuple  $\mathcal{A} = (Q, \Sigma, \delta, I, F, B)$  where  $Q$  is a finite set of *states*,  $I, F, B \subseteq Q$  are sets of *initial*, *final* and *Büchi-accepting* states, respectively, and  $\delta \subseteq Q \times \Sigma \times Q$  is the *transition relation*. A *run* of  $\mathcal{A}$  on a word  $w_0w_1\dots \in \Sigma^\infty$ ,  $w_i \in \Sigma$  is a maximal sequence of transitions  $t_0t_1\dots \in \delta^\infty$  with  $t_i = (q_i, w_i, q_{i+1})$  and  $q_0 \in I$ . It is *accepting* if it ends in a final state  $q_f \in F$  or visits a Büchi-accepting state  $q_b \in B$  infinitely often.  $\mathcal{A}$  *accepts*  $w$  if there is an accepting run of  $\mathcal{A}$  on  $w$  and the set of all accepted words is denoted  $\mathcal{L}(\mathcal{A})$ .

A *letter-to-letter transducer* is an  $\infty$ -automaton  $\mathcal{T} = (Q, \Sigma, \Gamma, \delta, I, F, B)$  where  $\Gamma$  is an additional *output alphabet* and  $\delta \subseteq Q \times \Sigma \times \Gamma \times Q$  is a *transition relation with output*. A word  $\gamma \in \Gamma^\infty$  is an *output* of  $\mathcal{T}$  if there is an accepting run of  $\mathcal{T}$  labeled by  $\gamma$ . For  $w \in \Sigma^\infty$  we denote  $\mathcal{T}(w) \subseteq \Gamma^\infty$  the set of possible outputs of  $\mathcal{T}$  when reading  $w$ .

**Data Words and Data Languages.** Let  $\Sigma$  be a finite alphabet,  $\Delta$  an infinite set of *data values* and  $A$  a finite set of *attributes*. A *multi-attributed data word* is a finite or infinite sequence  $w = w_0w_1\dots \in (\Sigma \times \Delta^A)^\infty$  of pairs  $w_i = (a_i, \mathbf{d}_i)$  of letters and *data valuations*  $\mathbf{d}_i : A \rightarrow \Delta$ . Given a valuation  $\mathbf{d} \in \Delta^A$  and a set of attributes  $X \subseteq A$  we denote by  $\mathbf{d}|_X$  the restriction of  $\mathbf{d}$  to  $X$ . We call  $\text{str}(w) := a_0a_1\dots \in \Sigma^\infty$  the *string projection* of  $w$ . The *X-class string* of  $w$  for a data valuation  $\mathbf{d} \in \Delta^X$  is the maximal projected subsequence  $\text{cl}(w, \mathbf{d}) :=$

$a_{i_0}a_{i_1}\dots \in \Sigma^\infty$  of  $w$  with  $0 \leq i_j \leq |w|$ ,  $i_j < i_{j+1}$  and  $\mathbf{d}_{i_j}|_X = \mathbf{d}$ . We use natural numbers  $1, 2, 3, \dots$  as representatives for arbitrary data values. For a data word  $w = (a_0, \mathbf{d}_0)(a_1, \mathbf{d}_1)\dots$  we also write  $(\overset{a_0}{\mathbf{d}_0}\overset{a_1}{\mathbf{d}_1}\dots)$ . For  $|A| = 1$  we call data words  $w \in (\Sigma \times \Delta^A)^\infty$  *single-attributed*. We may then omit the functional notation and use  $\Delta$  instead of  $\Delta^A$  if  $A$  is not essential, e. g., writing  $w \in (\Sigma \times \Delta)^\infty$ .

*Register Automata (RA).* A *register automaton* [7] over  $\Sigma$  and  $\Delta$  is a tuple  $\mathcal{R} = (Q, \Sigma, k, \delta, I, F, B)$  where  $Q$  is a finite set of states,  $I, F, B \subseteq Q$  are sets of initial, final and Büchi-accepting states, respectively,  $k \geq 1$  is the number of *registers* and  $\delta \subseteq Q \times 2^{[k]} \times 2^{[k]} \times \Sigma \times [k] \times Q$  is the transition relation. A *configuration* of  $\mathcal{R}$  is a pair  $(q, v)$  where  $q \in Q$  and  $v : [k] \rightarrow \Delta \cup \{\perp\}$  is a valuation of the registers. A *run* of  $\mathcal{R}$  on a single-attributed data word  $w = (a_0, d_0)(a_1, d_1)\dots \in (\Sigma \times \Delta)^\infty$  is a maximal sequence of configurations  $\rho = (q_0, v_0)(q_1, v_1)\dots$  s. t.  $q_0 \in I$  and for all  $0 \leq i < |w|$  there is a transition  $(q_i, R_i^-, R_i^+, a_i, x_i, q_{i+1}) \in \delta$  such that  $\forall r \in R_i^- v_i(r) = d_i, \forall r \in R_i^+ v_i(r) \neq d_i, v_{i+1}(x_i) = d_i$  and  $\forall r \neq x_i v_{i+1}(r) = v_i(r)$ . A run  $\rho$  of  $\mathcal{R}$  is *accepting* if it ends in a final state  $q \in F$  or it visits a Büchi-accepting state  $q \in B$  infinitely often. An RA accepts a single-attributed data word  $w$  if it has an accepting run on  $w$ .

**Multi-counter Systems.** A *reset multi-counter system (rMCS)* is a tuple  $\mathcal{M} = (Q, C, \delta, Q_0)$  where  $Q$  and  $C$  are finite sets of (*control*) *states* and *counters*, respectively,  $Q_0 \subseteq Q$  is the set of *initial states* and, for  $OP := \{\text{inc}, \text{dec}, \text{res}\}$ ,  $\delta \subseteq Q \times OP \times C \times Q$  is the *transition relation*. A *run* of  $\mathcal{M}$  is a sequence  $\rho \in Q_0 \times (OP \times C \times Q)^\infty$ , s. t. every subsequence  $(q, op, c, q')$  of  $\rho$ , with  $q, q' \in Q$ ,  $op \in OP$ ,  $c \in C$ , is an element of  $\delta$  and counters never become negative, i. e. there is an injection  $f_\rho : \mathbb{N} \rightarrow \mathbb{N}$  that maps every position  $i$  in  $\rho$  with  $(\rho_i, \rho_{i+1}) = (\text{dec}, c)$ , for  $c \in C$ , to a position  $j < i$  with  $(\rho_j, \rho_{j+1}) = (\text{inc}, c)$  and  $(\rho_k, \rho_{k+1}) \neq (\text{res}, c)$ , for all  $k$  with  $j < k < i$ . An *MCS* is an rMCS where the transition relation does not use the reset operation *res*.

### 3 Local Navigation in BD-LTL

The temporal logic BD-LTL is based on LTL. Linear-time properties are formulated using temporal operators to navigate along the positions of a word. This concept is extended analogously to data words by allowing for navigation along the occurrences of a data value. While the LTL operators express properties on the global structure of the word, independent of associated data values, navigation along the class strings of a word allows for expressing a local view, e. g., modeling the behaviour of a single process.

We now recall syntax and semantics of BD-LTL [12] and define two natural fragments  $\text{BD-LTL}^+$  and  $\text{BD-LTL}^-$  where local navigation is restricted to future and past operators, respectively. The satisfiability problem of BD-LTL is decidable. Yet, it is known to be as hard as reachability in Petri nets [12] and we show that satisfiability in our fragments is still 2EXPSpace-hard. The next section then

sharpens this result by developing a 2EXPSpace decision procedure based on restricted variants of data automata.

Let  $AP$  be a finite set of atomic propositions and  $A$  a finite set of attributes. The syntax of BD-LTL formulae consists of *position formulae*  $\varphi$  and *class formulae*  $\psi$ . It is defined by the following grammar where  $p \in AP$ ,  $x, y \in A$  and  $r \in \mathbb{Z}$ .

$$\begin{aligned}\varphi &::= p \mid \varphi \wedge \varphi \mid \neg \varphi \mid X \varphi \mid Y \varphi \mid \varphi U \varphi \mid \varphi S \varphi \mid C_x^r \psi \\ \psi &::= @x \mid \psi \wedge \psi \mid \neg \psi \mid X^= \psi \mid Y^= \psi \mid \psi U^= \psi \mid \psi S^= \psi \mid \varphi\end{aligned}$$

The semantics of BD-LTL position formulae  $\varphi$  is defined over models  $(w, i)$  consisting of an  $A$ -attributed data word  $w = (a_0, \mathbf{d}_0)(a_1, \mathbf{d}_1) \dots \in (\Sigma \times \Delta^A)^\infty$  over alphabet  $\Sigma = 2^{AP}$  and a position  $0 \leq i < |w|$ . Class formulae  $\psi$  are defined over models  $(w, i, d)$  containing an additional data value  $d \in \Delta$ . Boolean and LTL operators are defined as usual, ignoring the data values. For the semantics of the quantifier  $C_x^r$  and class formulae  $\psi$ , let  $\text{pos}_d(w) := \{i \mid 0 \leq i < |w|, \exists x \in A : \mathbf{d}_i(x) = d\}$  denote the set of positions  $i$  in  $w$  where some attribute has the value  $d \in \Delta$ . Then,

$$\begin{aligned}(w, i) &\models C_x^r \psi && \text{if } 0 \leq i + r < |w| \text{ and } (w, i + r, \mathbf{d}_i(x)) \models \psi, \\ (w, i, d) &\models \varphi && \text{if } (w, i) \models \varphi, \\ (w, i, d) &\models @x && \text{if } \mathbf{d}_i(x) = d, \\ (w, i, d) &\models X^= \psi && \text{if there is } j \in \text{pos}_d(w), j > i \\ &&& \text{and, for the smallest such } j, (w, j, d) \models \psi, \\ (w, i, d) &\models \psi_1 U^= \psi_2 && \text{if there is } j \in \text{pos}_d(w), j \geq i \text{ s.t. } (w, j, d) \models \psi_2 \\ &&& \text{and } \forall j' \in \text{pos}_d(w), j > j' \geq i : (w, j', d) \models \psi_1.\end{aligned}$$

The operators  $Y^=$  and  $S^=$  are furthermore defined as expected and  $(w, 0) \models \varphi$  is abbreviated  $w \models \varphi$ . We also use the abbreviations  $\top$  and  $F^= \varphi := \top U^= \varphi$ .

**Definition 1 (BD-LTL $^\pm$ ).** We define the following syntactical fragments: BD-LTL without operators  $X^=$  and  $U^=$  is called BD-LTL $^-$ . BD-LTL without operators  $Y^=$  and  $S^=$  is called BD-LTL $^+$ .

In [5], the *Logic of Repeating Values (LRV)* was introduced as an extension of LTL interpreted over multi-attributed data words. The additional operators are of the form  $x \approx X^r y$ ,  $x \approx \langle \varphi? \rangle y$  and  $x \not\approx \langle \varphi? \rangle y$ . The former expresses that the current value of attribute  $x$  must be equal to the value of attribute  $y$  at the position  $r$  steps ahead. Similarly, the latter two express that the value of  $x$  must eventually or never, respectively, be observed as the value of  $y$  at a position where, in addition, a formula  $\varphi$  holds. In  $\text{LRV}^\top$  only  $x \approx X^r y$  and  $x \approx \langle \top? \rangle y$  are allowed.  $x \approx X^r y$  and  $x \approx \langle \varphi? \rangle y$  can easily be translated into BD-LTL $^+$ :  $x \approx X^r y$  is equivalent to  $C_x^r @y$  and  $x \approx \langle \varphi? \rangle y$  is equivalent to  $C_x^0 X^= F^= (@y \wedge \varphi)$  [12]. On the contrary, LRV cannot express the operator  $X^=$ .

**Proposition 1.** BD-LTL $^+$  is strictly more expressive than  $\text{LRV}^\top$ .

The satisfiability problem of  $\text{LRV}^\top$  (and LRV) was shown to be 2EXPSpace-hard in [5] by encoding runs of so called chain automata using exponentially many

counters. The proof [5, Lemma 15] can easily be adapted to show that the variant of  $\text{LRV}^\top$  where past *instead* of future operators are used ( $x \approx Y^r y$ ,  $x \approx \langle \varphi? \rangle^{-1} y$ ) is also  $2\text{EXPSPACE}$ -hard and as  $\text{BD-LTL}^-$  subsumes this variant we obtain a lower bound for both of our fragments.

**Theorem 1 (Hardness).** *The satisfiability problems of  $\text{BD-LTL}^+$  and  $\text{BD-LTL}^-$  are  $2\text{EXPSPACE}$ -hard over both, finite and infinite data words.*

## 4 Satisfiability of $\text{BD-LTL}^\pm$ is $2\text{EXPSPACE}$ -complete

This section is dedicated to an exact characterization of  $\text{BD-LTL}^\pm$  satisfiability in terms of complexity. It also provides a basis for Section 6 that follows a similar structure but is technically more involved.

First, we formally define data automata and give restrictions that reflect the restrictions on our logic. They allow us to decide emptiness in  $\text{EXPSPACE}$ , as opposed to full data automata for which emptiness is as hard as reachability in Petri nets [1]. Second, we briefly recall the (exponential) translation from  $\text{BD-LTL}$  to data automata [19] and show that our logical restrictions indeed carry over to the restrictions on the automata side.

### 4.1 $\text{EXPSPACE}$ -variants of Data Automata

A *data automaton* ( $DA$ ) is a tuple  $\mathcal{D} = (\mathcal{A}, \mathcal{B})$  where the *base automaton*  $\mathcal{A} = (Q, \Sigma, \Gamma, \delta_{\mathcal{A}}, Q_0, F_{\mathcal{A}}, B_{\mathcal{A}})$  is a letter-to-letter transducer and the *class automaton*  $\mathcal{B} = (S, \Gamma, \delta_{\mathcal{B}}, I, F, B)$  is an  $\infty$ -automaton. A *memory function* of  $\mathcal{D}$  is a mapping  $f : \Delta \rightarrow S \cup \{\perp\}$  and we denote  $\mathfrak{F}$  the set of all memory functions. A *configuration* of  $\mathcal{D}$  is a tuple  $(q, f) \in Q \times \mathfrak{F}$  consisting of a base automaton state and a memory function. A *run* of  $\mathcal{D}$  on a single-attributed data word  $w = (a_0, d_0)(a_1, d_1) \dots \in (\Sigma \times \Delta)^\infty$  is a maximal sequence  $\rho = (q_0, f_0)(q_1, f_1) \dots \in (Q \times \mathfrak{F})^\infty$  such that  $q_0 \in Q_0$ ,  $\forall d \in \Delta : f_0(d) = \perp$  and for all consecutive positions  $i, i+1$  on  $\rho$  there is a transition  $(q_i, a_i, g, q_{i+1}) \in \delta_{\mathcal{A}}$  of the base automaton and a transition  $(s, g, s') \in \delta_{\mathcal{B}}$  of the class automaton such that (1)  $f_{i+1}(d_i) = s'$  and (2) either  $f_i(d_i) = s$ , or  $f_i(d_i) = \perp$  and  $s \in I$ , and (3)  $\forall d' \in \Delta, d' \neq d_i : f_i(d') = f_{i+1}(d')$ .

The run  $\rho$  is *accepting* if (I) it ends in a configuration  $(q, f)$  with  $q \in F_{\mathcal{A}}$  is final and  $f(\Delta) \cap S \subseteq F$ , or (II) there are infinitely many configurations  $(q, f_i)$  on  $\rho$  such that  $q \in B_{\mathcal{A}}$  is Büchi-accepting and for each data value  $d$  occurring *last* at some position  $i$  on  $w$  the state  $f_{i+1}(d) \in F$  is final and for each data value  $d'$  occurring *infinitely* often on  $w$  there are infinitely many positions  $j$  with  $d_j = d'$  and  $f_{j+1}(d') \in B$  is Büchi-accepting. The word  $w$  is accepted if there is an accepting run of  $\mathcal{D}$ .

Intuitively, the base transducer  $\mathcal{A}$  reads a letter  $a_i \in \Sigma$ , performs a transition and outputs its label  $g \in \Gamma$ . The memory function maintains an *instance* of the class automaton  $\mathcal{B}$  for every data value that occurred so far and spawns a new instance for a fresh data value. The (present or newly spawned) instance of  $\mathcal{B}$  that corresponds to the current data value  $d_i$ , reads  $g$  and performs a step.

For  $\mathcal{D}$  to accept,  $\mathcal{A}$  and every spawned instance of  $\mathcal{B}$  needs to accept by either terminating in a final state or visiting some Büchi-accepting state infinitely often.

**Definition 2 (Prefix- and suffix-closed DA).** A data automaton  $\mathcal{D} = (\mathcal{A}, \mathcal{B})$  is locally prefix-closed (pDA) if all states of the class automaton  $\mathcal{B}$  are final and Büchi-accepting. It is locally suffix-closed (sDA) if all states of  $\mathcal{B}$  are initial.

The construction to decide emptiness of DA given in [1] translates a DA into a multi-counter automaton (MCA) that maintains for every class automaton state the number of instances residing in it. That way, emptiness of DA reduces (for finite words) to reachability in MCA. Note that technical differences in the various notions of counter systems (e.g., MCA, MCS, VASS, Petri nets) are inessential here.

For pDA, where all class automaton states are final and Büchi-accepting, automaton instances can be dismissed in any state. The corresponding MCS thus allows for a random decrement of counters. Clearly, in such a *lossy* system the problem of reachability reduces to *coverability*. Regarding infinite words, *repeated coverability* is sufficient since every class automaton state is also Büchi-accepting. Both problems are in EXPSpace [20,21].

For an sDA we can decide if it accepts a *finite* word by reversing the automata and checking the resulting pDA for emptiness. In the rest of this section we address the remaining case of sDA emptiness wrt. *infinite* words obtaining the following result.

**Theorem 2.** *Emptiness of pDA and sDA over finite and infinite data words is decidable in EXPSpace.*

Let  $\mathcal{D} = (\mathcal{A}, \mathcal{B})$  be an sDA with  $\mathcal{A} = (Q, \Sigma, \Gamma, \delta_{\mathcal{A}}, Q_0, \emptyset, B_{\mathcal{A}})$  (we omit final states) and  $\mathcal{B} = (S, \Sigma, \delta, I, F, B)$ . Towards deciding emptiness of  $\mathcal{D}$ , we consider an accepting run  $\rho$  of  $\mathcal{D}$  and separate the finite from the infinite behaviour in terms of transitions  $t \in \delta$  of the class automaton: There is a position  $i$  on  $\rho$ , such that  $t$  is taken after  $i$  iff  $t$  is taken infinitely often on  $\rho$ .

The idea is now to guess (characteristics of) the configuration at this position and check that there is a finite run reaching the configuration and that starting from it there is an infinite accepting run. For the former, we construct an sDA that accepts a *finite* word iff the configuration is reachable. For the latter, we now have guaranteed infinite recurrence of all relevant transitions and can thereby reduce the problem to emptiness of an at most exponentially larger Büchi automaton.

For a set  $T \subseteq \delta$  of transitions of the class automaton  $\mathcal{B}$  and a state  $q \in Q$  of the base automaton  $\mathcal{A}$ , consider the following three properties.

- (A1) After taking any transition in  $T$ ,  $\mathcal{B}$  can eventually reach a final state from  $F$  or an accepting state from  $B$ , only by taking transitions from  $T$ .
- (A2) There is a sequence  $t_1 t_2 \dots \in T^\omega$  with  $t_i = (s_i, g_i, s'_i)$  in which each  $t \in T$  occurs infinitely often and  $g_1 g_2 \dots \in \Gamma^\omega$  is an output of  $\mathcal{A}$  starting in  $q$ .
- (A3) There is a reachable configuration  $(q, f)$  such that for all data values  $d \in \Delta$ , either (i) there is no corresponding instance of  $\mathcal{B}$  ( $f(d) = \perp$ ), or (ii) the corresponding instance of  $\mathcal{B}$  is in an accepting state ( $f(d) \in F$ ), or (iii) there is a transition  $(f(d), g, s) \in T$  for some  $s \in S$  and some  $g \in \Gamma$ .



**Lemma 1.** *The sDA  $\mathcal{D}$  accepts an infinite data word iff there are  $T \subseteq \delta$ ,  $q \in Q$  such that the properties (A1)–(A3) hold.*

( $\Rightarrow$ ) For an accepting run of  $\mathcal{D}$  take  $T$  to be the set of transitions of  $\mathcal{B}$  taken infinitely often on  $\rho$ . Let  $i$  be a position after which only transitions from  $T$  are taken and  $q_i \in Q$  the base automaton state at position  $i$ . If (A1) did not hold because of some transition  $t \in T$  then the instance of  $\mathcal{B}$  performing it after position  $i$  would reject. The suffix of  $\rho$  starting from  $i$  is a witness for (A2). The configuration  $(q_i, f_i) \in Q \times \mathfrak{F}$  at position  $i$  is a witness for (A3) as in particular all instances of  $\mathcal{B}$  that terminate before  $i$  accept.

( $\Leftarrow$ ) Given  $T$  and  $q$  we can construct an accepting run of  $\mathcal{D}$ . Property (A3) allows us to find a run to some configuration  $c \in Q \times \mathfrak{F}$  from which we are able to continue only using transitions from  $T$ .  $\mathcal{D}$  can then continue performing the sequence of transitions  $\tau \in T^\omega$  provided by (A2) since all states of  $\mathcal{B}$  are initial and hence new instances can be spawned in any state when needed. Now, (A1) and the fact that each transition in  $T$  occurs infinitely often on  $\tau$  guarantee that for any (non-terminated) instance of  $\mathcal{B}$  we can choose a subsequence of  $\tau$  that can be performed by  $\mathcal{B}$  in order to accept.

**Lemma 2.** *Given  $T \subseteq \delta$  and  $q \in Q$ , it is decidable in EXPSpace if properties (A1)–(A3) are satisfied.*

Verifying (A1) is a reachability problem in the finite graph of  $\mathcal{B}$  restricted to  $T$ . Further, we can build a Büchi automaton over  $\Gamma$  that is non-empty iff (A2) holds: In  $\mathcal{A}$ , take the outputs as inputs and remove all transitions with a label not occurring on any transition in  $T$ . For each transition  $(s, g, s') \in T$ , intersect the automaton with the property  $G F g$ . The size of the resulting Büchi automaton is at most  $c^{|Q|^2}$  for a constant  $c$ . Finally, (A3) can be verified by constructing the sDA  $\hat{\mathcal{D}} = (\hat{\mathcal{A}}, \hat{\mathcal{B}})$  with  $\hat{\mathcal{A}} = (Q, \Sigma, \Gamma, \delta_{\mathcal{A}}, Q_0, \{q\}, \emptyset)$  and  $\hat{\mathcal{B}} = (S, \Sigma, \delta, I, F \cup \bullet T)$ , where  $\bullet T := \{s \in S \mid \exists s' \in S, g \in \Gamma : (s, g, s') \in T\}$ , and checking it for emptiness in exponential space as above.

To conclude, using Lemma 1 and 2 we can check the sDA  $\mathcal{D}$  for emptiness wrt. infinite words by nondeterministically guessing a state  $q \in Q$  and a set of transitions  $T \subseteq \delta$  and verifying (A1)–(A3) in exponential space.

## 4.2 From BD-LTL $^\pm$ to Data Automata

By revisiting the construction given in [12] BD-LTL $^+$  and BD-LTL $^-$  can be translated into at most exponentially larger sDA and pDA, respectively.

The first step is to eliminate multiple attributes by translating a formula into a satisfiability-equivalent one over single-attributed data words. The basic idea is to encode  $A$ -attributed data words by using segments of length  $|A|$ , each representing a single position in the original word. The temporal operators are adjusted by offsets according to the segment length and positioning information within a fragment is encoded by additional propositions. The construction, given in detail in [19], only uses additional operators  $C_x^r$  and a BD-LTL $^\pm$  formula hence

stays in the respective fragment. Further, it is at most polynomially larger than the original one.

Second, the obtained formula  $\varphi$  is translated into a data automaton. The largest (absolute) value used by  $C_x^r$  operators in  $\varphi$  is denoted  $r_{max}$ . The set  $AP_\varphi$  of atomic propositions used by  $\varphi$  is extended by propositions  $p_j^\psi$  and  $=_j$  for each  $-r_{max} \leq j \leq r_{max}$  and subformula  $\psi$  of  $\varphi$ .  $\mathcal{A}_\varphi$  is supposed to check, that  $p_j^\psi$  holds at some position  $i$  iff  $\psi$  holds at position  $i+j$ . A proposition  $=_j$  is supposed to hold at position  $i$  iff position  $i+j$  carries the same data value. Checking all this and additionally that  $p_\varphi$  holds at the very first position,  $\mathcal{A}_\varphi$  accepts exactly the models of  $\varphi$ , up to the additional propositions.

Correct occurrence of propositions  $p_0^\psi$  where  $\psi$  is a position formula can be checked by the base automaton. In all other cases  $\psi$  can be assumed to be of the form  $C_x^r \psi$  where  $\psi$  is not a position formula. These formulae can be checked using the local automaton. The context information provided for  $r_{max}$  positions into the past as well as the future allows the local automaton to determine the effect of the  $C_x^r$  operator without any additional temporal navigation.

Propositions of the form  $p_j^\psi$  with  $j \neq 0$  can be handled by the base automaton. Note that in contrast to the construction given in [12] we require these propositions as a suffix or prefix closed local automaton can not keep track of this information itself. What remains is to verify the correct annotation by the propositions  $=_j$ . This can easily be done using a register automaton  $\mathcal{R}$  that maintains the frame of data values and verifies the propositions. While it is known that RA can be translated into DA it is not clear how to adapt the construction given in [12] for sDA and pDA.

**Lemma 3 (Simulating RA).** *Given an sDA or pDA  $\mathcal{D}$  and a register automaton  $\mathcal{R}$ , we can construct an sDA or pDA  $\mathcal{D}'$ , respectively, such that  $\mathcal{L}(\mathcal{D}) \cap \mathcal{L}(\mathcal{R}) = \emptyset$  iff  $\mathcal{L}(\mathcal{D}') = \emptyset$ .  $\mathcal{D}'$  is of polynomial size in the size of  $\mathcal{D}$  and  $\mathcal{R}$  and of exponential size in the number of registers of  $\mathcal{R}$ .*

The basic idea is to extend the alphabet with a new letter  $\$$ . At a position where a data value would have been stored in a register  $r$ , the class automaton changes its state and thereby marks the current data values as being stored in  $r$ . The transducer keeps track of all registers currently containing data values. Before “storing” another data value in an already occupied register  $r$ , the transducer performs an additional step accepting  $\$$  and demanding that one instance (the instance associated with the value currently “stored” in  $r$ ) changes its state, such that the data value is no longer marked as being stored in  $r$ . Both, a suffix or a prefix closed base automaton suffices to implement this approach.

By Lemma 3, we can perform all required checks using an sDA or pDA, respectively. Translating LTL formulae into word automata results in a state space that is at most exponential in the size of the formula and thus the construction gives an up to exponential overall blowup. Note that we assume a unary encoding for the offsets  $r$  in formulae  $C_x^r$ .

By Theorem 2, the translation proofs our completeness result for BD-LTL $^\pm$ .

**Theorem 3 (2EXPSpace-completeness).** *Satisfiability problems of  $BD\text{-}LTL^+$  and  $BD\text{-}LTL^-$  are 2EXPSpace-complete over finite and infinite data words.*

## 5 Ordered Navigation on Multi-attributed Data Words

As we have seen, multiple attributes do not actually enrich the models of  $BD\text{-}LTL$ . They can be eliminated due to the inability of  $BD\text{-}LTL$  to reason about their interdependencies. A natural extension is thus to allow for so-called *tuple navigation*, e. g., by adding an operator  $C_{(x,y)}^r$  binding a tuple instead of single values. Class operators such as  $X^=$  and  $S^=$  then navigate along the positions of a multi-attributed data word that carry both values. Unfortunately, it is well-known that such an extension leads to undecidability. For example, LRV is known to be undecidable when being extended by tuple navigation [5]. This implies undecidability of such an extension of  $BD\text{-}LTL^+$  and by similar arguments  $BD\text{-}LTL^-$ .

**Proposition 2.** *The satisfiability problem of  $BD\text{-}LTL^\pm$  with tuple navigation is undecidable.*

To overcome the restrictions of  $BD\text{-}LTL$  while maintaining decidability, at least for reasonable fragments, we define the logic  $ND\text{-}LTL$ .

**Definition 3 (ND-LTL).** *The logic Nested Data LTL ( $ND\text{-}LTL$ ) consists of  $BD\text{-}LTL$  formulae where the set of attributes  $A$  is enriched by a tree order relation  $\leq \subseteq A \times A$ . The fragments  $ND\text{-}LTL^+$  and  $ND\text{-}LTL^-$  are obtained by the same restrictions as for  $BD\text{-}LTL^+$  and  $BD\text{-}LTL^-$ , respectively.*

The quantifier  $C_x^r$  in  $ND\text{-}LTL$  binds not only the value of attribute  $x \in A$  but also the values of all smaller attributes. Class operators, such as  $U^=$ , then navigate according to this tuple of values respecting, however, the attribute order in the following sense.

For an attribute  $x \in A$ , with downward-closure  $x^\downarrow$  consisting of attributes  $x_1 < x_2 < \dots < x_n$ , a mapping  $\mathbf{d} \in \Delta^{x^\downarrow}$  induces a vector of data values  $(\mathbf{d}(x_1), \mathbf{d}(x_2), \dots, \mathbf{d}(x_n))$ . By  $\mathbf{d} \simeq \mathbf{d}'$  we denote that  $\mathbf{d}$  and  $\mathbf{d}'$  have the same such vector representation. Note, this can differ from the element-wise equality of the functions. Using this we define for a data word  $w \in (\Sigma \times \Delta^A)^\infty$  and  $\mathbf{d} \in \Delta^{x^\downarrow}$  the set  $\text{pos}_{\mathbf{d}}(w)$  of positions  $i$  in  $w$  where there is an attribute  $y \in A$  such that  $\mathbf{d} \simeq \mathbf{d}_i|_{y^\downarrow}$ .  $ND\text{-}LTL$  class formulae are interpreted over models  $(w, i, \mathbf{d})$  where  $i \in \mathbb{N}$ ,  $0 \leq i < |w|$ , is a position in  $w$  and  $\mathbf{d} \in \Delta^{x^\downarrow}$  for some  $x \in A$ . For position formulae  $\varphi$ ,  $x, y \in A$  and  $r \in \mathbb{Z}$ , we define the semantics of the  $C_x^r$  operator and class formulae  $\psi$  as follows.

$$\begin{aligned}
(w, i) &\models C_x^r \psi && \text{if } 0 < i + r < |w| \text{ and } (w, i + r, \mathbf{d}_i|_{x^\downarrow}) \models \psi, \\
(w, i, \mathbf{d}) &\models \varphi && \text{if } (w, i) \models \varphi, \\
(w, i, \mathbf{d}) &\models @x && \text{if } \mathbf{d}_i|_{x^\downarrow} \simeq \mathbf{d}, \\
(w, i, \mathbf{d}) &\models X^= \psi && \text{if there is } j \in \text{pos}_{\mathbf{d}}(w), j > i, \\
&&& \text{and, for the smallest such } j, (w, j, \mathbf{d}) \models \psi, \\
(w, i, \mathbf{d}) &\models \psi_1 U^= \psi_2 && \text{if } \exists_{j \in \text{pos}_{\mathbf{d}}(w), j \geq i} : (w, j, \mathbf{d}) \models \psi_2 \\
&&& \text{and } \forall_{j' \in \text{pos}_{\mathbf{d}}(w), j > j' \geq i} : (w, j', \mathbf{d}) \models \psi_1.
\end{aligned}$$

As before, the operators  $Y^=$  and  $S^=$  are defined as expected. The semantics of boolean and LTL operators in ND-LTL formulae remains as for BD-LTL.

**Lemma 4.** *For every rMCS  $\mathcal{M} = (Q, C, \delta, Q_0)$ , there is an ND-LTL<sup>-</sup> formula  $\Phi_{\mathcal{M}}$  over the set of propositions  $AP = Q \cup \{\text{inc}, \text{dec}, \text{res}\} \cup C$  and attributes  $A$  s. t.  $\Phi_{\mathcal{M}}$  is satisfiable iff there is a data word  $w \in (2^{AP} \times \Delta^A)^\omega$  where  $\text{str}(w) = \{p_0\}\{p_1\}\dots$  ( $p_i \in AP$ ) and  $p_0p_1\dots$  is a run in  $\mathcal{M}$ .*

Using a pair  $x_c > \hat{x}_c$  of attributes for each counter  $c \in C$ , a formula  $\bigwedge_{c \in C} G((\text{res} \wedge Xc) \rightarrow C_{\hat{x}_c}^0 \neg Y^= \top)$  can be used for specifying resets and  $\bigwedge_{c \in C} G((\text{dec} \wedge Xc) \rightarrow C_{x_c}^0 Y^= (\text{inc} \wedge Xc))$  assures non-negative counter values. It is clear that using a further constraint of the form  $Fq$  allows for expressing control state reachability in rMCS, being ACKERMANN-hard by results on lossy channel systems in [17]. Encoding such finite runs of an rMCS *backwards*, can be done analogously within the fragment ND-LTL<sup>+</sup>.

**Theorem 4 (ACK-hardness).** *Satisfiability of ND-LTL<sup>±</sup> is ACKERMANN-hard.*

Similarly,  $G F q$  expresses *repeated* control state reachability in rMCS, being undecidable due to results in [22]. Further, full ND-LTL is already undecidable over *finite* words. This can be shown by considering the formula  $\bigwedge_{c \in C} G((\text{inc} \wedge Xc) \rightarrow C_{x_c}^0 X^= (\text{dec} \wedge Xc))$  that ensures that for every incrementing operation, there is a following decrement on the same counter *before the next reset* on that counter. Thus, reset operations turn into *zero tests*, allowing to encode Minski machine computations where reachability is undecidable.

**Theorem 5 (Undecidability).** *Satisfiability of ND-LTL is undecidable over finite and infinite data words. Satisfiability of ND-LTL<sup>-</sup> is undecidable over infinite data words.*

## 6 Deciding Satisfiability of ND-LTL<sup>±</sup>

Having established undecidability and hardness results for ND-LTL we finally turn to decision procedures in this section. We complete our picture by decidability results for the remaining cases of ND-LTL<sup>-</sup> over finite words and ND-LTL<sup>+</sup> over finite and infinite words. The structure follows that of Section 4 and we provide the essential ideas for lifting the constructions as well as additional arguments where needed. To capture the notion of nesting in ND-LTL we extend data automata and again provide restrictions that carry over from the logic.

### 6.1 Nested Data Automata

We extend data automata to read multi-attributed data words by adding a class automaton for each attribute. The class automata are linearly ordered in the sense that the  $i$ -th class automaton reads refinements (subwords) of the input of the  $(i - 1)$ -th class automaton. That way they express a linear order on the

attributes which is, however, sufficient since we later show that ND-LTL formulae over a tree order can be translated into formulae over a linear order. For that reason, we only consider attribute sets  $[k] = \{1, \dots, k\}$  for  $k \in \mathbb{N}$ .

**Definition 4 (Nested data automaton).** A  $k$ -nested data automaton ( $k$ -NDA) is a  $(k+1)$ -tuple  $\mathcal{D} = (\mathcal{A}, \mathcal{B}_1, \dots, \mathcal{B}_k)$  where  $(\mathcal{A}, \mathcal{B}_i)$  is a data automaton for each  $i \in [k]$ .  $\mathcal{D}$  is called locally prefix-closed (pNDA) if each  $(\mathcal{A}, \mathcal{B}_i)$  is a pDA and it is called locally suffix-closed (sNDA) if each  $(\mathcal{A}, \mathcal{B}_i)$  is an sDA.

Let  $\mathcal{D} = (\mathcal{A}, \mathcal{B}_1, \dots, \mathcal{B}_k)$  be a  $k$ -NDA with  $\mathcal{A} = (Q, \Sigma, \Gamma, \delta_{\mathcal{A}}, Q_0, F_{\mathcal{A}}, B_{\mathcal{A}})$  and  $\mathcal{B}_i = (S_i, \Gamma, \delta_i, I_i, F_i, B_i)$ . A *configuration* of  $\mathcal{D}$  is a tuple  $c = (q, f_1, \dots, f_k) \in Q \times \mathfrak{F}_1 \times \dots \times \mathfrak{F}_k$  where  $\mathfrak{F}_i$  is the set of *memory functions*  $f : \Delta^{[i]} \rightarrow S_i \cup \{\perp\}$  (partially) mapping  $i$ -tuples of data values to states.

A *run* of  $\mathcal{D}$  on an  $[k]$ -attributed data word  $w = (a_0, \mathbf{d}_0)(a_1, \mathbf{d}_1) \dots \in (\Sigma \times \Delta^{[k]})^\infty$  is a maximal sequence  $\rho = (q_0, f_{1,0}, \dots, f_{k,0})(q_1, f_{1,1}, \dots, f_{k,1}) \dots$  of configurations where  $q_0 \in Q_0$ ,  $f_{i,0}(\Delta^{[i]}) = \{\perp\}$  and for each consecutive positions  $n, n+1$  on  $\rho$  there is a transition  $(q_n, a_n, g, q_{n+1}) \in \delta_{\mathcal{A}}$  for  $g \in \Gamma$  of the base automaton and a transition  $(s_i, g, s'_i) \in \delta_i$  for each class automaton  $\mathcal{B}_i$  such that (1)  $f_{i,n+1}(\mathbf{d}_n|_{[i]}) = s'_i$  and (2) either  $f_{i,n}(\mathbf{d}_n|_{[i]}) = s_i$ , or  $f_{i,n}(\mathbf{d}_n|_{[i]}) = \perp$  and  $s_i \in I_i$ , and (3)  $\forall \mathbf{d}' \in \Delta^{[i]}, \mathbf{d}' \neq \mathbf{d}_n|_{[i]} : f_{i,n}(\mathbf{d}') = f_{i,n+1}(\mathbf{d}')$ .

A run of  $\mathcal{D}$  on  $w$  is (finitely) *accepting* if it ends in a configuration  $(q, f_1, \dots, f_k)$  with  $q \in F_{\mathcal{A}}$  and  $\forall i \in [k] f_i(\Delta^{[i]}) \subseteq F_i \cup \{\perp\}$ . Moreover, it is accepting if there are infinitely many configurations  $(q, f_{1,n}, \dots, f_{k,n})$  on  $\rho$  such that  $q \in B_{\mathcal{A}}$  is Büchi-accepting and for each level  $i \in [k]$  and each data valuation  $\mathbf{d} \in \Delta^{[i]}$  there is either (I) *no* position  $m$  with  $\mathbf{d}_m|_{[i]} = \mathbf{d}$ , or (II) a *last* position  $m$  with  $\mathbf{d}_m|_{[i]} = \mathbf{d}$  and the state  $f_{i,m+1}(\mathbf{d}) \in F$  is final, or (III) there are *infinitely many* positions  $m$  where  $\mathbf{d}_m|_{[i]} = \mathbf{d}$  and  $f_{i,m+1}(\mathbf{d}) \in B_i$  is Büchi-accepting.

The idea of deciding emptiness of pNDA and sNDA is, again, to translate them into multi-counter systems, which this time will be nested. Similar notions of such nested systems can be found in [14, 11].

**Definition 5 ( $k$ -nMCS).** A  $k$ -nested multi-counter system ( $k$ -nMCS) is a tuple  $\mathcal{M} = (Q, \delta, I)$  with a finite set of states  $Q$ , a set of initial states  $I \subseteq Q$ , and a transition relation  $\delta \subseteq (\bigcup_{i \in [k]} Q^i) \times Q^k$ .

A *multiset* over a set  $S$  is a mapping  $m \in \mathbb{N}^S$ . For a  $k$ -nMCS  $\mathcal{M} = (Q, \delta, I)$ , the set of *configurations of level  $i$*  are defined inductively (from  $k$  to 0) as  $C_k = Q$  and  $C_{i-1} = Q \times \mathbb{N}^{C_i}$ . The set of *configurations* of  $\mathcal{M}$  is then  $C_{\mathcal{M}} = C_0$ . We can see an element of  $C_0$  as a term constructed over unary function symbols  $Q$ , constants  $Q$  and the binary operator  $+$ . The terms are considered modulo associativity and commutativity of the  $+$  operator which does not appear on the top level. For example  $q_0(q_1(q_3(q_5 + q_5 + q_6) + q_3(q_6 + q_6)) + q_1(q_3(q_6 + q_6) + q_3(q_6 + q_5 + q_5)) + q_2(q_7(q_8))) + q_2(q_7(q_8)))$  corresponds to  $(q_0, \{(q_1, \{(q_3, \{q_5 : 2, q_6 : 1\}) : 2, (q_3, \{q_6 : 2\}) : 2\}) : 1, (q_2, \{(q_7, \{q_8 : 1\}) : 1\}) : 2\})$ .

Now, the transition relation  $\rightarrow \subseteq C_{\mathcal{M}} \times C_{\mathcal{M}}$  on configurations can be easily defined as a rewrite rule. For  $((q_0, q_1, \dots, q_i), (q'_0, q'_1, q'_2, \dots, q'_k)) \in \delta$ , we have  $(q_0, X_1 + q_1(X_2 + \dots q_i(X_{i+1}) \dots)) \rightarrow (q'_0, X_1 + q'_1(X_2 + \dots q'_i(X_{i+1} +$

$q'_{i+1}(q'_{i+2} \dots q'_{k-1}(q'_k)))$  where  $X_i \in \mathbb{N}^{C_i}$ . As usual we denote by  $\rightarrow^*$  the reflexive and transitive closure of  $\rightarrow$ .

A *well-quasi-ordering* (WQO) on a set  $C$  is a pre-order  $\preceq$  such that, for any infinite sequence  $c_0, c_1, c_2, \dots$  there are  $i, j$  with  $i < j$  and  $c_i \preceq c_j$ . A WQO  $\preceq$  on a set  $C$  induces a WQO  $\preceq_m$  on multisets over  $C$  as follows. Let  $B = \{b_1, \dots, b_n\}$  and  $B' = \{b'_1, \dots, b'_{n'}\}$  two multisets over  $C$ . Then,  $B \preceq_m B'$  iff there is an injection  $h$  from  $[n]$  to  $[n']$  with  $b_i \preceq b'_{h(i)}$ . Let  $\preceq_k$  be the WQO = (equality relation) on the set of states  $Q$  of the  $k$ -nMCS  $\mathcal{M}$ . We iterate the construction and obtain a WQO  $\preceq_1$  on  $C_{\mathcal{M}}$ .

It can be easily seen that the transition relation  $\rightarrow$  of  $k$ -nMCS is *monotonic* wrt.  $\preceq_1$ , i.e., if  $c_1 \preceq_1 c_2$  and  $c_1 \rightarrow c_3$  then  $c_2 \rightarrow c_4$  for some  $c_4$  with  $c_2 \preceq_1 c_4$ . A  $k$ -nMCS is hence a *well-structured transition system* [16] and we directly obtain the following lemma.

**Lemma 5 (Coverability).** *Let  $\mathcal{M} = (Q, \delta, I)$  be a  $k$ -nMCS,  $c \in C_{\mathcal{M}}$  a configuration and  $q \in Q$  a state. The coverability problem of checking if there is a configuration  $c' \in C_{\mathcal{M}}$  with  $c \preceq c'$  such that  $(q, \emptyset) \rightarrow^* c'$ , is decidable.*

Given a  $k$ -NDA  $\mathcal{D} = (\mathcal{A}, \mathcal{B}_1, \dots, \mathcal{B}_k)$  where  $\mathcal{A} = (Q_0, \Sigma, \Gamma, \delta_0, I_0, F_0, \emptyset)$  and  $\mathcal{B}_i = (Q_i, \Gamma, \delta_i, I_i, F_i, \emptyset)$  for  $i \in [k]$  with disjoint sets of states and no Büchi-accepting states, we can construct a  $k$ -nMCS  $\mathcal{M}_{\mathcal{D}} = (\bigcup_{i=0}^k Q_i, \delta, I_0)$  as follows. Let  $((q_0, \dots, q_i), (q'_0, \dots, q'_k)) \in \delta$  for some  $0 \leq i \leq k$  if there are letters  $a \in \Sigma$  and  $g \in \Gamma$  such that there is a transition of the base automaton  $(q_0, a, g, q'_0) \in \delta_0$  and for all  $1 \leq j \leq i$  we have transitions  $(q_j, g, q'_j) \in \delta_j$  of the class automata and for all  $j$  with  $i < j \leq k$  there exists an initial state  $q''_j \in I_j$  such that  $(q''_j, g, q'_j) \in \delta_j$ . Then  $\mathcal{D}$  is empty iff a configuration can be reached in  $\mathcal{M}_{\mathcal{D}}$  containing only states from  $F := \bigcup_{i=0}^k F_i$ .

In case  $\mathcal{D}$  is a pNDA, all states of the class automata are final and the emptiness problem hence reduces to the coverability problem of  $k$ -nMCS. As above, if  $\mathcal{D}$  is an sNDA, we considering the reversal of the base and the class automata to obtain the case of pNDA (still without Büchi-accepting states). In the rest of this section we address the remaining case of checking if an sNDA with Büchi-accepting states accepts an infinite data word in order to obtain the following.

**Theorem 6.** *Emptiness of sNDA is decidable over finite and infinite data words. Emptiness of pNDA is decidable over finite data words.*

Now, let  $\mathcal{D} = (\mathcal{A}, \mathcal{B}_1, \dots, \mathcal{B}_k)$  be a  $k$ -sNDA where  $\mathcal{A} = (Q, \Sigma, \Gamma, \delta_{\mathcal{A}}, I_{\mathcal{A}}, \emptyset, B_{\mathcal{A}})$  and  $\mathcal{B}_i = (S_i, \Gamma, \delta_i, S_i, F_i, B_i)$ . For a configuration  $c = (q, f_1, \dots, f_k)$  of  $\mathcal{D}$ , a data valuation  $\mathbf{d} \in \Delta^{[1]}$  with  $f_1(\mathbf{d}) \neq \perp$  corresponds to an “active” instance of the class automaton  $\mathcal{B}_1$ . Consider the set  $m := \{\mathbf{d}' \in \Delta^{[i]} \mid i \in [k], f_i(\mathbf{d}') \neq \perp, \mathbf{d}'(1) = \mathbf{d}(1)\}$  of data valuations *depending* on  $\mathbf{d}$ . It is prefix-closed wrt. the linear order on  $[k]$  and can hence be considered as a *tree* with root  $\mathbf{d}$  (level 1). Define a labeling  $s : m \rightarrow \bigcup_{i \in [k]} S_i$  attaching to each node  $\mathbf{d}' \in \Delta^{[i]}$  (level  $i$ ) in  $m$  the current state of the corresponding class automaton instance, i.e.,  $s(\mathbf{d}') := f_i(\mathbf{d}')$ ,

and repeatedly delete all leaf nodes of  $m$  that are final states. Let  $M_c$  be the (finite) set of all such labeled trees  $(m, s)$  for a configuration  $c$ .

As done similar in Section 4, we characterize a configuration that splits the finite from the infinite behaviour on an accepting run of  $\mathcal{D}$ . For a set of transitions  $T \subseteq \delta_1$  of  $\mathcal{B}_1$ , a state  $q \in Q$  of  $\mathcal{A}$  and a finite set  $M$  of finite trees labeled by states from  $S_1 \cup \dots \cup S_k$ , consider the following properties.

- (B1) For all  $t_1 \in T$  there is a sequence  $t_1 t_2 \dots \in T^\infty$ ,  $t_i = (s_i, g_i, s'_i)$ , inducing an accepting run of  $\mathcal{B}_1$  and  $g_1 g_2 \dots \in \sqcup(\mathcal{L}(\mathcal{B}_2) \cap \sqcup(\dots \cap \sqcup(\mathcal{L}(\mathcal{B}_{k-1}) \cap \sqcup \mathcal{L}(\mathcal{B}_k)) \dots))$ .
- (B2) There is a sequence  $t_1 t_2 \dots \in T^\omega$  with  $t_i = (s_i, g_i, s'_i)$  in which each  $t \in T$  occurs infinitely often and  $g_1 g_2 \dots \in \Gamma^\omega$  is an output of  $\mathcal{A}$  starting in  $q$ .
- (B3) There is a reachable configuration  $c = (q, f_1, \dots, f_k)$  with  $M = M_c$  such that for all  $i \in [k]$  and all  $\mathbf{d} \in \Delta^{[i]}$  either (i)  $f_i(\mathbf{d}) = \perp$  (there is no corresponding instance), or (ii)  $\forall \mathbf{d}' \in \Delta^{[i]} \text{ s.t. } \mathbf{d}'|_{[i]} = \mathbf{d} \forall j \geq i : f_j(\mathbf{d}'|_{[j]}) \in F_j$  (the corresponding instance and all instances depending on it are in a final state), or (iii)  $\exists_{g \in \Gamma, s' \in S} : (f_1(\mathbf{d}|_{[1]}, g, s') \in T$  (there is a transition applicable to the corresponding instance of  $\mathcal{B}_1$ ).
- (B4) For each tree  $(m, s) \in M$  there is a second labeling  $\gamma : m \rightarrow \Gamma^\infty$  such that, for the root  $r \in m$ , the label  $\gamma(r)$  is accepted by  $\mathcal{B}_1$  restricted to  $T$  when starting in state  $s(r) \in S_1$  and for all nodes  $v \in m$  on a level  $i > 1$  (i)  $\gamma(v)$  is accepted by  $\mathcal{B}_i$  starting in state  $s(v) \in S_i$  and (ii)  $\gamma(v)$  must be a shuffle of the labels of the direct children of  $v$  and a (possibly infinite) number of words from the shuffle set  $\sqcup(\mathcal{L}(\mathcal{B}_{i+1}) \dots \cap \sqcup(\mathcal{L}(\mathcal{B}_{k-1}) \cap \sqcup \mathcal{L}(\mathcal{B}_k)) \dots)$ .

**Lemma 6.** *The sNDA  $\mathcal{D}$  accepts an infinite data word iff there are  $T \subseteq \delta_1$ ,  $q \in Q$  and a set  $M$  of finite trees labeled by states from  $S_1 \cup \dots \cup S_k$  s.t. properties (B1)–(B4) hold.*

For a complete proof see Appendix D.1. It is based on similar arguments as Lemma 1. The new aspect is to schedule class automaton instances on higher levels consistently.

**Lemma 7.** *For  $T \subseteq \delta_1$  and  $q \in Q$  we can decide if there is a set  $M$  of finite trees labeled by states from  $S_1 \cup \dots \cup S_k$  such that the properties (B1)–(B4) hold.*

Given  $T$  we verify (B2) as above by constructing and analyzing a Büchi automaton. We now sketch the procedure to compute the candidates  $M$  that satisfy (B3). We construct a  $k$ -sNDA  $\tilde{\mathcal{D}} = (\tilde{\mathcal{A}}, \tilde{\mathcal{B}}_1, \dots, \tilde{\mathcal{B}}_k)$ , without Büchi-accepting states, from  $\mathcal{D}$  by taking  $q$  as only final state in  $\tilde{\mathcal{A}}$ . In each step,  $\tilde{\mathcal{A}}$  guesses whether the currently active instance of  $\tilde{\mathcal{B}}_1$  performs its last step entering a source state  $s$  of some transition  $(s, g, s') \in T$ . In that case it marks the current output by some flag.  $\tilde{\mathcal{B}}_1$  simulates  $\mathcal{B}_1$  and verifies that  $\tilde{\mathcal{A}}$  guessed correctly. Each other class automaton  $\tilde{\mathcal{B}}_i$  ( $i > 1$ ) simulates  $\mathcal{B}_i$ . Upon reading the flag it moves to an accepting copy of the state they would have moved to otherwise.

The configurations in which  $\tilde{\mathcal{D}}$  can accept are exactly those configurations reachable by  $\mathcal{D}$  that satisfy (B3). We apply the standard saturation algorithm

for well-structured transition systems where constraints are propagated from the target control state backwards along the edges of the nMCS. After its termination, the algorithm computed the minimal preconditions for reaching a the target state. On a reversed sNDA, this can be understood as a forward propagation computing minimal post-conditions. In this case the target state is  $q$  and the minimal post conditions characterize the minimal configurations  $(q, f_1, \dots, f_k)$  that can be reached. Here, minimal means with the smallest number of instances of some class automaton. The post-conditions hence give us all minimal sets  $M$  when reaching  $q$ . These are the (finitely many) candidates for (B4) since if none of those satisfies the properties any larger one will not either.

Now, for testing the candidates  $M$  to comply (B4) and  $T$  to satisfy (B1), the essential idea is to let the shuffle requirements be checked by a  $(k - 1)$ -sNDA built by modifying the components of  $\mathcal{D}$ . Such an automaton is constructed for each  $(m, s) \in M$  and each  $t \in T$ , respectively, and can, by induction, be checked for emptiness.

## 6.2 From ND-LTL to NDA

The translation from ND-LTL $^\pm$  to sNDA and pNDA, respectively, follows closely the one for BD-LTL in Section 4.2. For an ND-LTL formula over arbitrarily ordered attributes, a word has at every position a tree of attributes with  $f$  maximal paths of length of at most  $k$ . The first step is to translate this formula to a formula over the linearly order set of attributes  $[k]$  and encode each position of such a word by a segment of length  $f$ , where each position within a segment corresponds to a maximal path in the tree order  $(A, \leq)$ . This step is crucial as NDA only navigate according to linearly ordered attributes.

For translating the obtained formula  $\varphi$  into an NDA, the set  $AP_\varphi$  of atomic propositions used by  $\varphi$  is extended by propositions  $p_j^\psi$  and  $=_j^x$  for each  $-r_{\max} \leq j \leq r_{\max}$  and subformula  $\psi$  and attribute  $x$  of  $\varphi$ , where  $r_{\max}$  denotes the largest (absolut) value used by  $C_x^r$  operators. As before positional formulae can be checked by the base automaton. Class formulae of the form  $C_x^r \psi$  can be handled by the local automaton corresponding to attribute  $x$ . Propositions  $=_j^x$  are checked separately for each attribute  $x$  by adapting the construction used for Lemma 3.

Now, together with Theorem 6, we obtain a decision procedure for ND-LTL $^\pm$ .

**Theorem 7.** *Satisfiability of ND-LTL $^+$  is decidable over finite and infinite data words. Satisfiability of ND-LTL $^-$  is decidable over finite data words.*

**Corollary 1.** *Emptiness of pNDA wrt. infinite data words is undecidable.*

## References

1. Bojanczyk, M., David, C., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data words. *ACM Trans. Comput. Log.* **12**(4) (2011) 27
2. Schwentick, T., Zeume, T.: Two-variable logic with two order relations. *Logical Methods in Computer Science* **8**(1) (2012)



3. Demri, S., Lazic, R., Nowak, D.: On the freeze quantifier in constraint LTL: Decidability and complexity. *Inf. Comput.* **205**(1) (2007) 2–24
4. Demri, S., Lazic, R.: LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.* **10**(3) (2009)
5. Demri, S., Figueira, D., Praveen, M.: Reasoning about data repetitions with counter systems. In: *LICS*, IEEE Computer Society (2013) 33–42
6. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.* **5**(3) (2004) 403–435
7. Kaminski, M., Francez, N.: Finite-memory automata. *Theor. Comput. Sci.* **134**(2) (1994) 329–363
8. Bouyer, P., Petit, A., Thérien, D.: An algebraic approach to data languages and timed languages. *Inf. Comput.* **182**(2) (2003) 137–162
9. Björklund, H., Schwentick, T.: On notions of regularity for data languages. *Theor. Comput. Sci.* **411**(4-5) (2010) 702–715
10. Tzevelekos, N., Grigore, R.: History-register automata. In Pfenning, F., ed.: *FoSSaCS*. Volume 7794 of *LNCS.*, Springer (2013) 17–33
11. Björklund, H., Bojanczyk, M.: Shuffle expressions and words with nested data. In Kucera, L., Kucera, A., eds.: *MFCS*. Volume 4708 of *LNCS.*, Springer (2007) 750–761
12. Kara, A., Schwentick, T., Zeume, T.: Temporal logics on words with multiple data values. In Lodaya, K., Mahajan, M., eds.: *FSTTCS*. Volume 8 of *LIPIcs*. (2010) 481–492
13. Leroux, J.: Vector addition system reachability problem: a short self-contained proof. In Ball, T., Sagiv, M., eds.: *POPL*, ACM (2011) 307–316
14. Lomazova, I.A., Schnoebelen, P.: Some decidability results for nested Petri nets. In Bjørner, D., Broy, M., Zamulin, A.V., eds.: *Ershov Memorial Conference*. Volume 1755 of *LNCS.*, Springer (1999) 208–220
15. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! *Theor. Comput. Sci.* **256**(1-2) (2001) 63–92
16. Abdulla, P.A.: Well (and better) quasi-ordered transition systems. *Bulletin of Symbolic Logic* **16**(4) (2010) 457–515
17. Schnoebelen, P.: Revisiting ackermann-hardness for lossy counter machines and reset Petri nets. In Hliněný, P., Kucera, A., eds.: *MFCS*. Volume 6281 of *LNCS.*, Springer (2010) 616–628
18. Demri, S., D’Souza, D., Gascon, R.: Temporal logics of repeating values. *J. Log. Comput.* **22**(5) (2012) 1059–1096
19. Kara, A., Schwentick, T., Zeume, T.: Temporal logics on words with multiple data values. *CoRR* **abs/1010.1139** (2010)
20. Rackoff, C.: The covering and boundedness problems for vector addition systems. *Theor. Comput. Sci.* **6** (1978) 223–231
21. Habermehl, P.: On the complexity of the linear-time  $\mu$ -calculus for Petri nets. In Azéma, P., Balbo, G., eds.: *ICATPN*. Volume 1248 of *LNCS.*, Springer (1997) 102–116
22. Bouajjani, A., Mayr, R.: Model checking lossy vector addition systems. In Meinel, C., Tison, S., eds.: *STACS*. Volume 1563 of *LNCS.*, Springer (1999) 323–333

## A Local Navigation in BD-LTL

A straight forward lemma that is used implicitly in the constructions is that the classes of pDA and sDA are closed under union and intersection.

**Lemma 8 (Closure).** *Suffix- and prefix-closed data automata are closed under union and intersection.*

**Proof.** For the intersection of two sDA or two pDA, carry out the usual product construction the base and class automata separately. An automaton accepting the union can be constructed by letting the base automaton perform a non-deterministic choice of one of automata and output a flag on the first letter indicating that choice. The class automaton then simulates the class automaton of the data automaton that was chosen. It is easy to see that these constructions result in an sDA or pDA if the two original automata were both an sDA or a pDA, respectively.

## B Satisfiability of BD-LTL<sup>±</sup> is 2EXPSpace-complete

### B.1 EXPSpace-variants of Data Automata

For showing that an sNDA  $\mathcal{D}$  can be checked for emptiness wrt. infinite words we use Lemma 1 for which we provide more detailed proof here. Recall that  $\mathcal{D} = (\mathcal{A}, \mathcal{B})$  with  $\mathcal{A} = (Q, \Sigma, \Gamma, \delta_{\mathcal{A}}, Q_0, \emptyset, B_{\mathcal{A}})$  and  $\mathcal{B} = (S, \Sigma, \delta, I, F, B)$ .

**Lemma 1 (necessity).** Assume  $\mathcal{D}$  has an accepting run  $\rho \in (Q \times \mathfrak{F})^\omega$  on some word  $w \in (\Sigma \times \Delta)^\omega$ . Let  $T \subseteq \delta$  be the set of transitions of the class automaton  $\mathcal{B}$  taken infinitely often by  $\rho$ . Then, there exists some position  $i$  on  $\rho$  with  $\rho_i = (q, f)$  and, in the suffix  $\rho_i \rho_{i+1} \dots$ , only transitions from  $T$  are taken by (any instance of) the class automaton  $\mathcal{B}$ . If there were a transition  $t \in T$  violating property (A1), some instance of  $\mathcal{B}$  would reject since  $t$  is taken eventually. As this is not the case, property (A1) holds.

Second, we record that the configuration  $(q, f)$  meets the requirements of property (A3). We assumed that only transitions from  $T$  are taken after position  $i$  carrying  $(q, f)$ . Hence, if there were some data value  $d \in \Delta$  violating (A3), the corresponding instance of  $\mathcal{B}$  would not accept since it can neither take any transition anymore nor is it in an accepting state.

Third, the suffix  $\rho_i \rho_{i+1} \dots$  of  $\rho$  is a witness that (A2) is satisfied.

**Lemma 1 (sufficiency).** To see that the opposite direction also holds, consider a run  $\rho$  of  $\mathcal{D}$  that leads to  $(q, f)$  and continues by applying the the sequence of transitions  $\tau = t_1 t_2 \dots$  provided by (A2).  $\rho$  is accepting since  $\mathcal{A}$ , starting in  $q$ , can correctly continue to move and produce the labels of the transitions while meeting its Büchi condition. Further, each active instance of  $\mathcal{B}$  is identified by some data value  $d \in \Delta$  with  $f(d) \neq \perp$ . For those with  $f(d) \in F$ , we can just

consider them discontinued, hence they accept. A crucial point is that we can indeed always apply the transition sequence  $\tau$  since even if there is no active instance of  $\mathcal{B}$  that allows for a particular transition  $t = (s, g, s')$ , we can always spawn a new instance of  $\mathcal{B}$  in  $s \in S$  since  $\mathcal{B}$  is *suffix closed*, i.e. all states are initial.

Property (A1) guarantees, that all instances of  $\mathcal{B}$  are accepting when considering the following scheduling approach. Put all (finitely many) active instances in some queue of „temporarily completed“ instances. Whenever a new instance is created it is appended to the queue. Take the first instance of the queue. The last transition taken by it, there is a finite sequence of transitions that leads  $\mathcal{B}$  to a final or Büchi state. This sequence is guaranteed to occur as a (possibly scattered) subsequence in  $\tau$  and we just wait for the next suitable transition to occur while dispatching all intermediate transitions to other (existing or new) instances of  $\mathcal{B}$ . Upon reaching a final or Büchi state we can dismiss it or append it to the queue, respectively. All instances are scheduled infinitely often or terminate in some final state. This way we can construct an accepted data word where the data value at each position corresponds to the active instance of  $\mathcal{B}$ .

## B.2 From BD-LTL<sup>±</sup> to Data Automata

**Proof (Simulating RA (Lemma 3)).** Intuitively, the DA  $\mathcal{D}'$  that is supposed to be empty if and only if the intersection of the DA  $\mathcal{D}$  and the RA  $\mathcal{R}$  is empty, simulates both simultaneously. The base automaton of  $\mathcal{D}'$  simulates the base automaton of  $\mathcal{D}$  and the finite control of  $\mathcal{R}$ . In its state it keeps track of the registers currently in use. A register becomes marked as in use, when a data value is stored. Before a new data value can be stored, the register has to be marked as free. Therefore, the base automaton takes a special transition marked by the new input symbol  $\$$  and guesses the data value currently stored in the register. The base automaton encodes in its output when it marks a register as in use or as free, and which registers have to be compared for equality or inequality with respect to the current data value. The class automaton of  $\mathcal{D}'$  simulates the class automaton of  $\mathcal{D}$  and keeps track of the registers the data value associated with the class projection it reads is currently stored in. Thus it can verify that the current data value is actually stored in the register that the base automaton expects.

For sake of simplicity, we do not consider Büchi accepting states for the following construction. They can be easily handled using the usual product construction for Büchi automata. Let  $\mathcal{D} = (\mathcal{A}, \mathcal{B})$  be a data automaton  $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, \Gamma, \delta_{\mathcal{A}}, I_{\mathcal{A}}, F_{\mathcal{A}}, \emptyset)$  and  $\mathcal{B} = (Q_{\mathcal{B}}, \Gamma, \delta_{\mathcal{B}}, I_{\mathcal{B}}, F_{\mathcal{B}}, \emptyset)$ . Let  $\mathcal{R} = (Q_{\mathcal{R}}, \Sigma, k, \delta_{\mathcal{R}}, I_{\mathcal{R}}, F_{\mathcal{R}}, \emptyset)$  be a register automaton. We can define  $\mathcal{D}' = (\mathcal{A}', \mathcal{B}')$  with base automaton  $\mathcal{A}' = (Q_{\mathcal{A}'}, \Sigma', \Gamma', \delta_{\mathcal{A}'}, I_{\mathcal{A}'}, F_{\mathcal{A}'})$  and class automaton  $\mathcal{B}' = (Q_{\mathcal{B}'}, \Gamma', \delta_{\mathcal{B}'}, I_{\mathcal{B}'}, F_{\mathcal{B}'})$  with  $\Sigma' = \Sigma \cup \{\$\}$  and  $\Gamma' = \Gamma \times 2^{[k]} \times 2^{[k]} \times [k] \cup [k]$  by:

- $Q_{\mathcal{A}'} = Q_{\mathcal{A}} \times Q_{\mathcal{R}} \times 2^{[k]}$
- $I_{\mathcal{A}'} = I_{\mathcal{A}} \times I_{\mathcal{R}} \times \{\emptyset\}$
- $F_{\mathcal{A}'} = F_{\mathcal{A}} \times F_{\mathcal{R}} \times 2^{[k]}$

- $((q_A, q_R, R), a, (\gamma, R_-, R_+, r), (q'_A, q'_R, R \cup \{r\})) \in \delta_{A'}$  iff:
  - $(q_R, R_-, R_+, a, r, q'_R) \in \delta_R$
  - $(q_A, a, \gamma, q'_A) \in \delta_A$
  - $r \notin R$
- $((q_A, q_R, R), \$, r, (q_A, q_R, R \setminus \{r\})) \in \delta_{A'}$  iff  $r \in R$
- $Q_{B'} = Q_Q \times 2^{[k]}$
- $I_{B'} = I_B \times \{\emptyset\}$
- $F_{B'} = F_B \times \{\emptyset\}$
- $((q_B, R), (\gamma, R_-, R_+, r), (q'_B, R \cup \{r\})) \in \delta_{B'}$  iff
  - $(q_B, \gamma, q'_B) \in \delta_B$
  - $R_- \in R$
  - $R \cap R_- = \emptyset$
  - $r \notin R$
- $((q_B, R), (\gamma, R_-, R_+, r), (q_B, R \setminus \{r\})) \in \delta_{B'}$  iff  $r \in R$

In every step for every non-empty register  $r$  only one instance of the class automaton may be in a state, denoting that the associated data value is stored in  $r$ . This is guaranteed by the construction by assuring, using the base automaton, that a new data value is only stored once a register has been freed and that it is only freed if it contains a value, and by assuring using the class automaton, that the value at the storing position is the same as at the next following freeing position. It can be observed, that when the class automaton sees a store action, it will eventually see a free action (and no store or free action in between), or vice versa when it sees a free action, it has already seen a store action. Hence, the DA can be turned into a prefix or suffix closed DA by turning all states either into final (and Büchi accepting) or initial states.

## C Ordered Navigation on Multi-attributed Data Words

**Proof (Tuple Navigation (Proposition 2)).** BD-LTL<sup>+</sup> subsumes  $LRV^\top$  which is known to be undecidable already over finite words when being extended by tuple navigation [5]. Let  $\varphi$  be a BD-LTL<sup>+</sup> formula over finite words. We can construct a satisfiability-equivalent formula  $\hat{\varphi}$  by replacing all future operators by past operators and vice versa and replacing every subformula  $\psi$  by  $\neg \$ \wedge \psi$ . Now  $F(\hat{\varphi} \wedge X G \$)$  is satisfiable if and only if  $\varphi$  is. Thus, the satisfiability problem of BD-LTL<sup>+</sup> tuple navigation is also undecidable.

**Proof (Encoding rMCS (Lemma 4)).** For the set of attributes we take  $A := \{x_c, \hat{x}_c \mid c \in C\}$ , i. e., two attributes for each counter of  $\mathcal{M}$ . Let the tree order  $\leq$  be defined s. t. attributes for different counters are incomparable and  $x_c > \hat{x}_c$  for all  $c \in C$ . We let  $\Phi_{\mathcal{M}}$  be the conjunction composed of the following formulae.

- $\Phi_\delta$  specifies that a data word has to have the shape of a run according to the finite relation  $\delta$ , e. g., the correct order of states, followed by an operator, a counter and, again, a state. This can be done by only using plain LTL formulae.

- $\Phi_{\text{res}} := \bigwedge_{c \in C} G((\text{res} \wedge Xc) \rightarrow C_{\hat{x}_c}^0 \neg Y \neg \top)$  specifies, that whenever a reset happens on any counter  $c$ , the current data value has never been seen before (in any of the top-level attributes  $\hat{x}_{c'}$ , including that for  $c$ ).
- $\Phi_{\text{dec}} := \bigwedge_{c \in C} G((\text{dec} \wedge Xc) \rightarrow C_{x_c}^0 Y \neg (\text{inc} \wedge Xc))$  says, that for each decrement operation, the previous position with the same data must carrying an increment operation on the same counter. Note that in conjunction with  $\Phi_{\text{res}}$ , there must not be a reset on the same counter in between since using  $x_c$  in the formula means comparing the values of both attributes,  $x_c$  and  $\hat{x}_c$ .

## D Deciding Satisfiability of ND-LTL $^\pm$

### D.1 Nested Data Automata

In the following we provide the proof for Lemma 6, stating that it is necessary and sufficient to find a set  $T \in \delta_1$ , a state  $q \in Q$  and a set  $M$  of labeled trees such that the conditions (B1)–(B4) hold, in order to decide that an  $k$ -sNDA  $\mathcal{D}$  to be decide non-empty.

Let  $\mathcal{D} = (\mathcal{A}, \mathcal{B}_1, \dots, \mathcal{B}_k)$  with

- $\mathcal{A} = (Q, \Sigma, \Gamma, \delta_{\mathcal{A}}, I_{\mathcal{A}}, \emptyset, B_{\mathcal{A}})$  and
- $\mathcal{B}_i = (S_i, \Gamma, \delta_i, S_i, F_i, B_i)$  for  $i \in [k]$ .

**Proof (Lemma 6, necessity).** Assume  $\mathcal{D}$  has an accepting run  $\rho \in (Q \times \mathfrak{F}_1 \times \dots \times \mathfrak{F}_k)^\omega$  on some infinite data word  $w = w_0 w_1 \dots \in (\Sigma \times \Delta^{[k]})^\omega$  with  $w_i = (a_i, \mathbf{d}_i)$ . The run  $\rho$  induces a sequence  $\tau \in \delta_1^\omega$  of transitions of  $\mathcal{B}_1$  that are performed between consecutive configurations. This need not to be a run in  $\mathcal{B}_1$  but it is by definition a shuffle of runs of  $\mathcal{B}_1$  and since  $\rho$  is accepting each of these *class* runs is accepting. Let  $T \subseteq \delta_1$  be the maximal set of transitions of  $\mathcal{B}_1$  that occur infinitely often in  $\tau$ .

There is a position  $N$  in  $\tau$  (and  $\rho$ ) after which only transitions from  $T$  occur. Any transition  $t_1 \in T$  occurs eventually on the suffix of  $\tau$  starting at this position. The class run that  $t_1$  belongs to is accepting and since all states in  $\mathcal{B}_1$  are initial also each suffix of that class run is an accepting run of  $\mathcal{B}_1$ . In particular, the suffix starting with  $t_1$ .

This is our witness for (B1): It consists only of transitions from  $T$ . Further, by the definition runs of NDA and the fact that all class automata states are initial, the sequence  $\gamma$  of labels along that run of  $\mathcal{B}_1$  is a shuffle of words accepted by  $\mathcal{B}_2$  which in turn are shuffles of words accepted by  $\mathcal{B}_3$  and so on. Intuitively, the instances of  $\mathcal{B}_2$  reading parts of  $\gamma$  are those where the corresponding data valuation is an extension of the data valuation corresponding the instance of  $\mathcal{B}_1$  reading  $\gamma$ .

We choose the state  $q$  to be that occurring in the configuration  $\rho_N = (q, f_1, \dots, f_k)$  at position  $N$  in  $\rho$ . Recall, that after  $N$  only transitions from  $T$  occur in  $\tau$ . The the suffix  $\tau_{N+1} \tau_{N+2} \dots$  of  $\tau$  is then the witness for (B2). Since  $\rho$  is accepting,  $\mathcal{A}$ , starting in  $q$  can output the sequence of labels corresponding to  $\tau_{N+1} \tau_{N+2} \dots$ .

Third, we record that  $\rho_N = (q, f_1, \dots, f_k)$  is a reachable configuration for which property (B3) holds. Property (B4) states that at the configuration  $\rho_N$  fixed for property (B3), each active instance of a class automaton  $\mathcal{B}_x$  and all instances depending on it (present and spawned at later positions) must accept when continuing with  $\rho$ . This is the case since  $\rho$  is accepting.

**Proof (Lemma 6, sufficiency).** We have seen that if an sNDA  $\mathcal{D}$  is non-empty, there is a reachable configuration  $c = (q, f_1, \dots, f_k)$  inducing a set  $M_c$  of trees and a set of transitions  $T$  such that properties (1)-(4) hold.

To see that the opposite direction also holds, consider a run  $\rho$  of some sNDA  $\mathcal{D} = (\mathcal{A}, \mathcal{B}_1, \dots, \mathcal{B}_k)$  that leads to the configuration  $c = (q, f_1, \dots, f_k)$  and continues by applying the sequence of transitions  $\tau \in T^\omega$  provided by (B2). The base automaton  $\mathcal{A}$  accepts since starting in  $q$  it can correctly continue to move and produce the labels of the transitions while meeting its Büchi condition. This run of  $\mathcal{A}$  already yields the string projection for data word  $w$  accepted by  $\mathcal{D}$  and it remains to argue that it is possible to correctly choose data values for  $w$ .

Up to reaching configuration  $c$ , property (B3) assures that there is a choice of data values such that all class projections are accepted by the corresponding class automaton, except for those represented in  $M_c$ . By (B4) we can “complete” these instances as follows. For each root of a tree  $(m, s) \in M_c$  we know that there is a sequence of transitions  $\tau_{(m,s)} \in T^\omega$  of  $\mathcal{B}_1$  such that  $\mathcal{B}_1$  accepts. Further, that run induces a word  $\gamma_{(m,s)} \in \Gamma^\omega$  and it is possible to decompose it into subsequences that can be assigned to the depending active instances (represented by the other nodes in  $m$ ) or to newly spawned instances of class automata of the respective level such that all those instances accept. Note that a spawning a new instance on some level  $x$ , which corresponds to a fresh data value in the model, implies spawning a new dependent instance for each level  $y > x$  as the data words we consider do not allow for missing data values. Property (B4) assures acceptance of those as the words annotated to leaf nodes that are not on the bottom level must still be a shuffle of words accepted by the class automata  $\mathcal{B}_{y'}$ ,  $k \geq y' > y$ , below.

We “execute” each sequence  $\tau_{(m,s)} = t_1 t_2 \dots$  by choosing the first occurrence of  $t_1$  in the global sequence  $\tau$  and devote it to the instance represented by the root of  $(m, s)$  by labeling the position of model  $w$  where  $t_1$  occurs with the corresponding data value. We continuously choose the respective next occurrence of the transitions  $t_2 \dots$  and proceed analogously. This is possible since (B2) guarantees that in  $\tau$  each transition occurs always eventually.

Having treated these “obligations” imposed by  $M_c$ , we can now fill the remaining positions where data values are missing in  $w$  as follows. Choose fresh data values for the first free position. For the transition  $t \in T$  taken at this position in  $\tau$ , property (B1) provides a sequence assuring that the new instances spawned will accept. This sequence is distributed over the global sequence  $\tau$  the same way as the sequences  $\tau_{(m,s)}$  above. Note that we can always choose a fresh data value, no matter which transition needs to be taken since the corresponding class automaton is suffix-closed and can thus be started in any of its states. The model  $w$  constructed that way is a witness that  $\mathcal{D}$  is non-empty.

## D.2 From ND-LTL to NDA

For translating ND-LTL formulae to NDA we first reduce formulae over arbitrary tree-orders  $(A, \leq)$  to a satisfiability-equivalent formula over the *linearly* ordered set  $[k]$ . We provide this translation in detail in terms of the following lemma.

**Lemma 9.** *Let  $(A, \leq)$  be a tree order of attributes with tree-height  $k$ . For each ND-LTL $^\pm$  formula over  $(A, \leq)$  we can construct a satisfiability-equivalent ND-LTL $^\pm$  formula over attributes  $[k]$  with the linear order on the natural numbers.*

**Proof.** Let  $f$  be the number of maximal paths in the graph of  $(A, \leq)$ . We enumerate them and let  $f(x)$ , for  $x \in A$ , denote the lowest number  $i$  such that  $x$  occurs on the  $i$ -th such path. Further, let  $0 \leq h(x) < k$  be the level of  $x$  in the corresponding tree, i.e. the distance to its root. We let  $A' := \{y_0, \dots, y_{k-1}\}$  be linearly ordered by  $y_0 \leq' y_1 \leq' \dots \leq' y_{k-1}$ .

The idea is now, similar to the construction in Section 4.2, to encode words over the tree order  $(A, \leq)$  into words over the linear order of attributes  $[k]$ , where we have segments of length  $f$  corresponding to single positions in the original data word. Each position within a certain frame corresponds to a maximal path in the tree order  $(A, \leq)$ .

We transform a given ND-LTL $^\pm$  formula  $\Phi$  over attributes  $(A, \leq)$  and propositions  $AP$  into an ND-LTL $^\pm$  formula  $\hat{\Phi}$  over attributes  $[k]$  and propositions  $AP' := AP \dot{\cup} \{p_1, \dots, p_f\}$ . The additional propositions are intended to mark the positions in each frame (i.e. modulo  $f$ ), which can easily be expressed by an LTL formula  $\Phi_e$ .

We can assume that  $\Phi$  has a normal form where each  $X^=$ ,  $Y^=$ ,  $U^=$  and  $S^=$  formula is directly preceded by  $C_x^r$ . This is due to the equalities

$$\begin{aligned} C_x^r \neg \varphi &\equiv (\neg C_x^r \varphi) \wedge \beta_r, \\ C_x^r (\varphi \wedge \psi) &\equiv (C_x^r \varphi) \wedge (C_x^r \psi), \\ C_x^r X \varphi &\equiv X^r \varphi, \\ C_x^r Y \varphi &\equiv Y^r \varphi, \\ C_x^r (\varphi U \psi) &\equiv X^r (\varphi U \psi), \\ C_x^r (\varphi S \psi) &\equiv X^r (\varphi S \psi), \\ \varphi U^= \psi &\equiv \psi \vee (\varphi \wedge X^= (\alpha(\varphi) U^= \alpha(\psi))), \\ \varphi S^= \psi &\equiv \psi \vee (\varphi \wedge Y^= (\alpha(\varphi) S^= \alpha(\psi))), \\ X^= \varphi &\equiv X^= \alpha(\varphi), \text{ and} \\ Y^= \varphi &\equiv Y^= \alpha(\varphi), \end{aligned}$$

where  $\beta_r$  is an LTL formula checking that it is possible to move  $r$  steps,  $\alpha(\varphi) := \bigwedge_{x \in A} (@x \rightarrow C_x^0 \varphi)$  and, for  $r < 0$ ,  $X^r$  denotes  $Y^{-r}$  and vice versa.

Then,  $\hat{\Phi} := \Phi_e \wedge t(\Phi)$  where we define  $t(\Phi)$  as follows.

$$\begin{aligned}
t(p) &= p \\
t(\neg\varphi) &= \neg t(\varphi) \\
t(\varphi \wedge \psi) &= t(\varphi) \wedge t(\psi) \\
t(\varphi \vee \psi) &= t(\varphi) \vee t(\psi) \\
t(X\varphi) &= X^f t(\varphi) \\
t(Y\varphi) &= Y^f t(\varphi) \\
t(\varphi U \psi) &= t(\varphi) U t(\psi) \\
t(\varphi S \psi) &= t(\varphi) S t(\psi) \\
t(C_x^r(\varphi U^\equiv \psi)) &= \bigwedge_{j=1}^f p_j \rightarrow X^{f(x)-j} C_{y_h(x)}^{fr}(\varphi U^\equiv \psi) \\
t(C_x^r(\varphi S^\equiv \psi)) &= \bigwedge_{j=1}^f p_j \rightarrow X^{f(x)-j} C_{y_h(x)}^{fr}(\varphi S^\equiv \psi) \\
t(C_x^r X^\equiv \varphi) &= \bigwedge_{j=1}^f p_j \rightarrow X^{f(x)-j} C_{y_h(x)}^{fr-f(x)+f}(X^\equiv \varphi) \\
t(C_x^r Y^\equiv \varphi) &= \bigwedge_{j=1}^f p_j \rightarrow X^{f(x)-j} C_{y_h(x)}^{fr-f(x)}(Y^\equiv \varphi)
\end{aligned}$$

The correctness of the transformation can easily be seen by considering the underlying invariant, that the constructed formulae are evaluated equally on every position of a particular frame. Note that under the transformation, any ND-LTL $^\pm$  formula stays in its respective fragment.

For the final translation to NDA we rely on the fact that we can check the additional propositions  $=_j^x$ . This can be easily done using a register automaton. To obtain the type of automaton we need, such as pNDA or sNDA, we can then use the following lemma.

**Lemma 10.** *For the linearly ordered set of attributes  $[k]$  and a natural number  $r \in \mathbb{N}$ , let  $\Sigma$  be a finite alphabet that includes a set  $P = \{=_j^x \mid -r \leq j \leq r, x \in A\} \subseteq \Sigma$  of dedicated propositions.*

*Let  $\mathcal{D}$  be a  $k$ -nested NDA over  $A$ ,  $\Sigma$  and a data domain  $\Delta$  and  $E \subseteq (\Sigma \times \Delta^{[k]})^\infty$  be the language of all  $[k]$ -attributed data words  $w = (\mathbf{a}_0 \mathbf{a}_1 \dots)$ ,  $\mathbf{a}_i \in \Sigma$ ,  $\mathbf{d}_i \in \Delta^{[k]}$ , such that*

$$\forall_{0 \leq i < |w|} =_j^x \in \mathbf{a}_i \text{ iff } \mathbf{d}_i|_{[x]} = \mathbf{d}_{i+j}|_{[x]}.$$

*We can construct a  $k$ -nested NDA  $\mathcal{D}'$  such that  $\mathcal{D}' = \emptyset$  iff  $\mathcal{D} \cap E = \emptyset$ . Further, if  $\mathcal{D}$  is an sNDA or an pNDA then  $\mathcal{D}$  is an sNDA or an pNDA, respectively.*



**Proof.** For each attribute  $x$  we can, in the same fashion as in Section 4.2, build a dedicated register automaton that reads tuples of data values  $(d_1, \dots, d_k)$  and checks the correct annotation of propositions  $=_j^x$  wrt. to the values in the  $x$ -th level. Following the construction from Lemma 3, we can construct an NDA that simultaneously simulates  $\mathcal{D}$  and all of the register automata for each level  $x$ . For each register automaton, the construction only involves the base automaton and the respective  $x$ -th class automaton. As argued above, the single class automata remain suffix- or prefix-closed.